THE UNIVERSITY of York
DEPARTMENT OF COMPUTER SCIENCE

USER MANUAL

NATHANIEL THWAITES-MCGOWAN

ABBREVIATIONS

Whilst this document is aimed at readers with a Computer Science background, this list of acronyms is intended to help those without the required depth of hardware knowledge.

CPU	Central Processing Unit
DJNZ	Decrement and jump if not zero (instruction used for looping)
EPROM	Erasable-Programmable Read Only Memory (reusable ROM)
GUI	Graphical User Interface
IC	Integrated Circuit (or microchip)
NMI	Non-Maskable Interrupt
PIO	Parallel Input / Output – Z80 family microchip providing I/O
RAM	Random Access Memory (volatile, temporary storage)
ROM	Read Only Memory (non-volatile, permanent storage)

CONTENTS

1 INTRODUCTION 4

2 GETTING STARTED 5

Downloading ZIM 5

Simulating execution 5

3 MEMORY CONFIGURATION 7

ROM 7

RAM 8

Memory-Mapped I/O 9

Memory-Mapped Serial I/O 9

4 PERIPHERAL DEVICES 10

Discrete I/O 10

Discrete Serial I/O 11

Peripheral Screen Components 11

5 DEBUGGING FEATURES 13

Watch Points 13

Break Points 14

6 LOGGING 15

Instruction Tracing 15

I/O Logging 15

7 SAVING YOUR CONFIGURATION 16

8 COMPATIBILITY 17

1 Introduction

The Department of Computer Science
The University of York
York, YO10 5DD
United Kingdom

24th February 2004

Dear User,

Welcome to ZIM – The Z80 Machine Simulator.

ZIM is a fully featured Z80 CPU Simulation tool designed as an aide for students studying modules involving the Z80 assembly language.

ZIM has been designed with users in mind – incorporating an advanced GUI, allowing the visualization of code simulation in real-time.

Extensive debugging features are also provided:

- Advanced break-points and watch-points,
- Instruction Tracing
- I/O logging
- The ability to request interrupts from the GUI

and much much more…

ZIM has been designed to be intuitive, so those with knowledge of a Z80 system should be able to make full use of all the features immediately. For the less informed reader, this manual will lead you through the basics, and introduce you to the more advanced features.

All comments and queries are gratefully received.

N. Thwaites-McGowan
njtm100@cs.york.ac.uk

2 Getting Started

THIS SECTION COVERS GETTING HOLD OF AN AUTHENTIC COPY OF ZIM, INSTALLING AND RUNNING IT ON YOUR PC, AND SIMULATING SIMPLE PROGRAMS.

DOWNLOADING AND INSTALLING ZIM

ZIM is a Java application, and is available as a download for Windows, Linux and the MacOS platforms.

Visit http://www-students.cs.york.ac.uk/~njtm100/ for a copy of ZIM. You will also require the Java Runtime Environment, which can be obtained direct from sun at http://java.sun.com

By far the best way of running ZIM is through the Java Web Start facility. Having installed the Java Runtime Environment, you will also have automatically installed Java Web Start. This enables you to simply click on the "Launch ZIM" link on the website to run the application. Downloading, version updating and local storage for when you want to run the application offline, are all handled automatically by JWS.

SIMULATING EXECUTION

When you've successfully downloaded a copy of ZIM, or accessed it through Web Start, you're ready to begin simulating code.

ZIM's initial state is that of a fairly standard Z80 Machine. It has a RAM module (at 0x4000 – 0x8000), a memory-mapped input device (at 0x8000 – 0xC000), and a memory-mapped output device (at 0xC000 – 0xF000). The remaining memory area is reserved for a ROM device containing your program code.

Use the *File* menu and *Open* | *Program* to select your Z80 binary executable. It will be configured as a ROM device starting at 0x0000. The contents of the ROM are viewable by clicking on the *ROM* button corresponding to your device in the memory-map (a second memory-viewer is created displaying the ROM data).

Pressing STEP on the CPU Control window will step through simulation of the code. Pressing START will execute the code at full speed.

RESET sets the SP to 0xFFFF and all other registers to 0x0000. The interrupt most is set to Mode 0, and interrupts are disabled.

Interrupts can be requested through the use of the INT (interrupt) and NMI (non-maskable interrupt) buttons on the *CPU Control* component. Some interrupt modes require data to be passed to the data bus, in this case ZIM will request the data-bus values by way of a dialog box.

Register values and memory location values are viewable using the *CPU Registers* component (pictured below) and the *Memory Viewer* component respectively.

Values in these components are updated in real-time at the end of every instruction – when the simulated machine is in a consistent state.

The *Stack Viewer* keeps track of the number of items on the stack and illustrates them in a LIFO structure.

The State Setup component allows the value of any register or memory-location to be altered in-between stepping the CPU – allowing jumping to the end of loops, or correction of the results of coding errors.

Components are hidden from view by clicking on the cross in the top right hand corner. They can also be minimized. In order to make a hidden component visible again use the *View* menu.

3 Memory Configuration

THIS SECTION COVERS ADVANCED CONFIGURATION OF THE MEMORY-MAP ASSOCIATED WITH THE SIMULATED Z80 CPU.

ROM – READ ONLY MEMORY

ROM is traditionally the storage medium of choice for program code. It is non-volatile and read-only – the code is protected. ZIM has the ability to simulate ROM devices within the memory map – protecting the contents from any memory-writing. Data can be loaded into a simulated ROM device in two ways.

File | Open This method was illustrated in *Section One* and creates a ROM chip the size of your code with a base-address of 0x0000. This is by far the easiest way to load program-code into the simulator.

Memory Map If not already visible this tool may be activated using the *View | Memory Map* menu option.

- Click on *ADD* – top right hand side.
- An *Add Memory Space* dialog box appears.
- Choose ROM from the *Type* drop-down selector.
- *Click to Choose* your program file
- Enter a base-address (e.g. 0000), taking care to ensure that your new ROM chip won't overlap with any other chips already configured.
- Click *ADD*.

Assuming no overlapping occurred, and that all values are within the range of the memory-space (0000 – FFFF), the device will be created.

As a helpful debugging aide, if you're making frequent changes to your binary code, a refresh option is provided. After adding a ROM module, the *ADD* button changes to read *Refresh*. A single click of *Remove* – next to your ROM in the memory-map, and then a click of *Refresh* will reload the modified code from the file into memory.

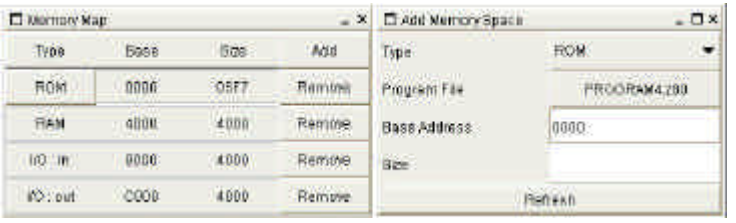
RAM – RANDOM ACCESS MEMORY

RAM is volatile memory – temporary storage space. Before your program code can write data into memory, an area of RAM must be configured. If your code attempts to write to an area of the memory-space that is not configured as RAM, the data will not be successfully saved, and won't be available during any subsequent read of that same location.

Memory Map If not already visible this tool may be activated using the *View | Memory Map* menu option.

- Click on *ADD* – top right hand side.
- An *Add Memory Space* dialog box appears.
- Choose RAM from the *Type* drop-down selector.
- Enter a base-address (e.g. 0000), taking care to ensure that your new RAM chip won't overlap with any other chips already configured.
- Enter a size (e.g. 3FFF), once again taking care to ensure that no chips overlap.
- Click *ADD*.

Assuming no overlapping occurred, and that all values are within the range of the memory-space (0000 – FFFF), the device will be created.



MEMORY-MAPPED I/O

In addition to ROM and RAM devices, memory-mapped I/O devices can also be configured. These represent peripheral devices that can be accessed through standard memory instructions. They are created in exactly the same manner as RAM devices – *see above.* Whilst memory-mapped I/O devices can encompass many memory-locations, each memory-location within one will *act in exactly the same manner*.

i.e. If a device active from 4000– 5FFF then
a write to 4015 will have the same effect as a write to 5F3A

Input Input devices can be read from by the Z80, but not written to. Any data written to an input device (much like ROM) is ignored. The user can feed data into an input device by mapping it to a peripheral component on the screen (see *Section Four)*.

Output Output devices are the opposite– they can be written to by the Z80, but data cannot be read back. The user can view the last byte output, using a periphal component on the screen (see *Section Four*).

MEMORY-MAPPED SERIAL I/O

Serial I/O devices are bi-directional memory-mapped I/O devices, which exhibit similar properties to RAM devices.

Serial I/O Serial I/O devices map any data output to COM1 (Windows only), and any read requests are served with data inbound on COM1. Program execution halts to wait for data on COM1 if a read request is made, and no data is available.

Data is queued, so if 5 bytes are sent to COM1 and only 2 are read using the Z80, 3 will remain in the queue for the next three read operations.

4 Peripheral Devices

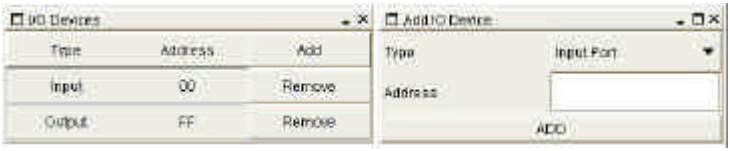
THIS SECTION COVERS ADVANCED CONFIGURATION OF PERIPHERAL DEVICES THAT CAN BE ATTACHED TO THE SIMULATED Z80 CPU.

DISCRETE I/O

Discrete I/O allows peripheral devices to be mapped, not into the memory-space, but into a separate I/O space. The Z80 I/O space has 256 locations (00 – FF), and devices are accessed through special instructions – IN and OUT.

Input Input devices can be read from by the Z80, but not written to. Any data written to an input device (much like ROM) is ignored. The user can feed data into an input device by mapping it to a peripheral component on the screen

Output Output devices are the opposite – they can be written to by the Z80, but data cannot be read back. The user can view the last byte output, using a periphal component on the screen .



DISCRETE SERIAL I/O

Serial I/O devices are bi-directional I/O devices, which exhibit similar properties to RAM devices.

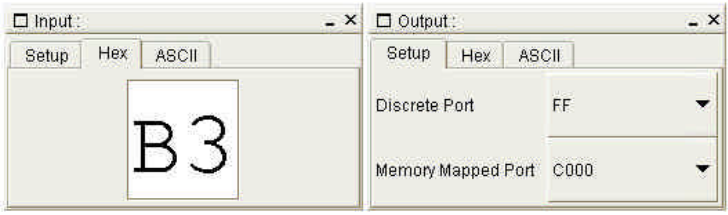
Serial I/O Serial I/O devices map any data output to COM1 (Windows only), and any read requests are served data inbound on COM1. Program execution halts to wait for data on COM1 if a read request is made, and no data is available. Data is also queued, so if 5 bytes are sent to COM1 and only 2 are read using the Z80, 3 will remain in the queue for the next three read operations.

PERIPHERAL SCREEN COMPONENTS

As explained above, an input device can only be read from by the Z80, not written to. For such a device to be useful, the user must be able to provide the data which is read by the Z80. Peripheral Screen Components provide that feature – allowing the user to input data or view output data – interfacing with either configured memory-mapped or discrete I/O devices.

A periphal screen component is created using the *File* | *New* menu option, and choosing between an input or an output device.

The binding between the screen component and the actual configured device is set in the *Setup* tab on the component itself. Use one of the two drop-down menus (one for discrete devices, one for memory-mapped) to choose the device to bind to. Once this is completed, any one of the tabs along the top of the component may be chosen, allowing different representations of the data to be used (ASCII, Hex, Lights etc).



When inputting data, the enter key must be pressed after placing the data in the input box. This updates the cached value ready for the next read.

LCD DISPLAY SIMULATOR

ZIM includes a simulated LM016, which is a 2 line x 16 character LCD display by Fujitsu. It can be found on the SBC computers used in MCP.

The display has two registers used to interface with it:

The Control Register	0xB8
The Data Regster	0xB9

These need to be configured as Discrete I/O devices. When they are available as devices, the *File | New | LCD Display* command will create a simulated LCD Screen bound to those devices. *No other Peripheral Device Component should be bound to view 0xB8 or 0xB9 when the LCD Screen is active – they will prevent the LCD from reading the outputs effectively.*

Instructions giving the command set for the LM016 are available widely on the internet or on the MCP course website.

5 Debugging Features

THIS SECTION COVERS THE BREAK AND WATCH-POINT FEATURES AVAILABLE FOR DEBUGGING Z80 PROGRAM CODE.

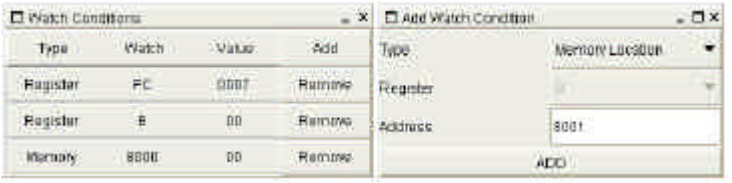
WATCH POINTS

Watch-points provide the ability to keep a watch on various attributes of the simulated machine. They are configured via the watch-point manager.

Watches may be placed on

- 8-bit Registers
- 16-bit Registers
- Memory Locations

As the value of the attributes on watch changes, the watch-point manager is updated, and displays the new data. This enables quick and easy visulation of a large range of attributes in the same screen area.



If, for example a loop is being executed – or perhaps a loop within a loop – the invarient and varient values may be spread out between registers and memory locations. If all relevant locations and registers are placed on watch, the execution and progression of the loop become much more visible.

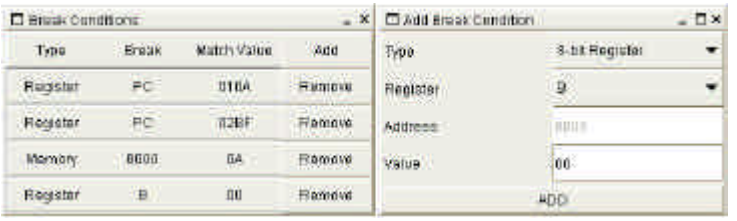
BREAK POINTS

Break-points operate in much the same way as watch-points, except instead of displaying a current value for all attributes, a match value is configured.

At the end of every instruction execution this match-value is compared to the curent value. If a successful match is detected, CPU execution is paused and the user is informed.

Break-points may be set on values in

- 8-bit Registers
- 16-bit Registers
- Memory Locations



Break-points are especially useful for executing code in fast-execution mode up to a certain point (matching against the PC) or until a certain condition occurs (matching against any register or memory location).

Halting execution at the end of a DJNZ loop is possible for example, by creating a break-point matching the register B to 0x00 (or 0x01 if you want to step through the last interation).

6 Logging

THIS SECTION COVERS THE ABILITY TO LOG BOTH EXECUTED INSTRUCTIONS AND I/O OVER A PERIOD OF TIME TO A LOG FILE.

INSTRUCTION TRACING

Instruction tracing enables a log file to be built containing a list of all the instructions executed during the logging period.

File | Save | Set Instruction Trace File
This command sets the instruction trace file and enables intruction tracing. *Log | Trace* subsequently enables or disables instruction trace logging. The log file contains an instruction on each line.

File | Save | Set Message Log File
This command sets the message log file – which logs the contents of the *Messages* pane – and enables message logging. *Log | CPU Messages* subsequently enables or disables message logging. The log contains all the instructions executed along with the state of the CPU after each one. FILE.

I/O LOGGING

I/O logging saves the values of all configured outputs, with an entry being created *every time any of the outputs changes value.*

File | Save | Set I/O Log File
This command sets the I/O log file. *Log | I/O* subsequently enables or disables I/O logging. The log contains a column containing the number of tStates since execution began (representing time – 1 tState is equal to 250ns) and one column representing every configured output device.

7 Saving your Configuration

THIS SECTION COVERS THE ABILITY TO SAVE YOUR MEMORY-MAP AND I/O DEVICE CONFIGURATION TO A FILE, AND RESTORE IT AT A LATER DATE.

SAVING YOUR CONFIGURATION

The facility to save the current configuration of ZIM covers

- Memory-mapped devices (but not ROM)
- Discrete I/O devices

Use *File* | *Save* | *Current Configuration*

Choose a directory and filename, and click *Save*.

LOADING A PREVIOUS CONFIGURATION

Use *File* | *Open* | *Previous Configuration*

Every device but ROM devices will be restored as before. The screen layout however, and log file settings will remain set as default.

8 Compatibility

THIS SECTION COVERS THE INSTRUCTION SET COMPATIBILITY OF ZIM.

Z80

ZIM simulates *every documented instruction* included in the Zilog Z80 CPU.

Z180

ZIM simulates the following instructions

- MLT rr Multiply
- IN0 r, (n) Input
- OUT0 (n), r Output

The input and output instructions behave as the input and output instructions on a standard Z80.

(The Z180 has three I/O device spaces, two internal, and one external. IN0 and OUT0 are used for interfacing with the internal I/O spaces. As ZIM does not support internal I/O spaces, IN0 and OUT0 will interface with *any* device configured as a Discrete I/O device.)

THE Z80 REGISTER ARCHITECTURE

	Main Register Set		Alternate Register Set	
	Accumulator	Flag Register	Accumulator	Flag Register
	A	F	A′	F′
General Purpose Registers	B	C	B′	C′
	D	E	D′	E′
	H	L	H′	L′
Special Purpose Registers	Interrupt Vector	Memory Refresh		
	I	R		
	Index Register IX			
	Index Register IY			
	Stack Pointer SP			
	Program Counter PC			

THE ASCII TABLE

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

ZIM – THE Z80 MACHINE SIMULATOR

Copyright © 2003-2004 by N. Thwaites-McGowan

The right of Nathaniel Thwaites-McGowan to be identified as the Author of the work as been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved.